SCHOOL OF COMPUTING
UNIVERSITY OF UTAH

Formal Analysis of GPU Programs

with Atomics via

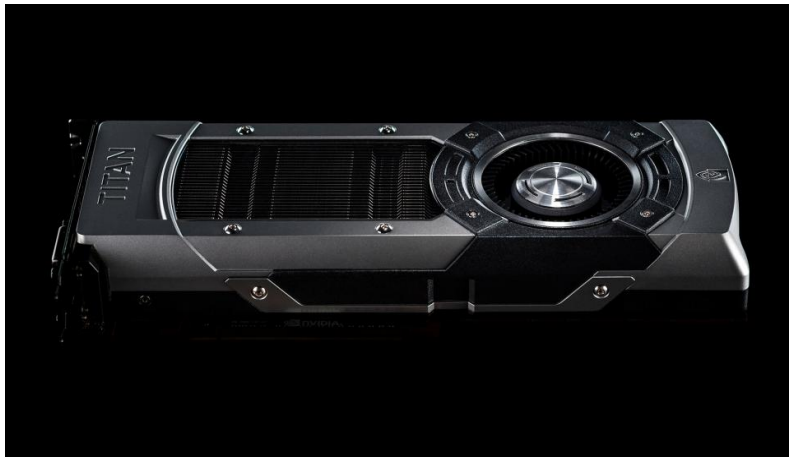
Conflict-Directed Delay-Bounding

Wei-Fan Chiang

Joint work with Zvonimir Rakamarić, Ganesh Gopalakrishnan, and Guodong Li

Motivation

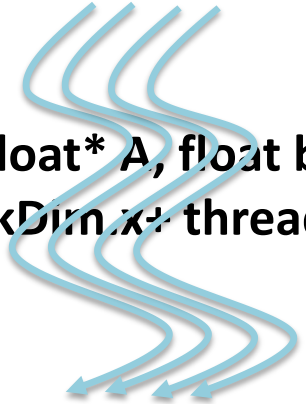
- Use of GPUs growing!
  - Extreme-Scale Computing, Mobile Devices,…
  - High Compute Rates, Parallelism
- This work : *how to design correct GPU programs?*



Contrast between CPUs and GPUs

Example: Increment Array Elements

Fine-grained threads scheduled to run like this:
tid = 0, 1, 2, 3, …



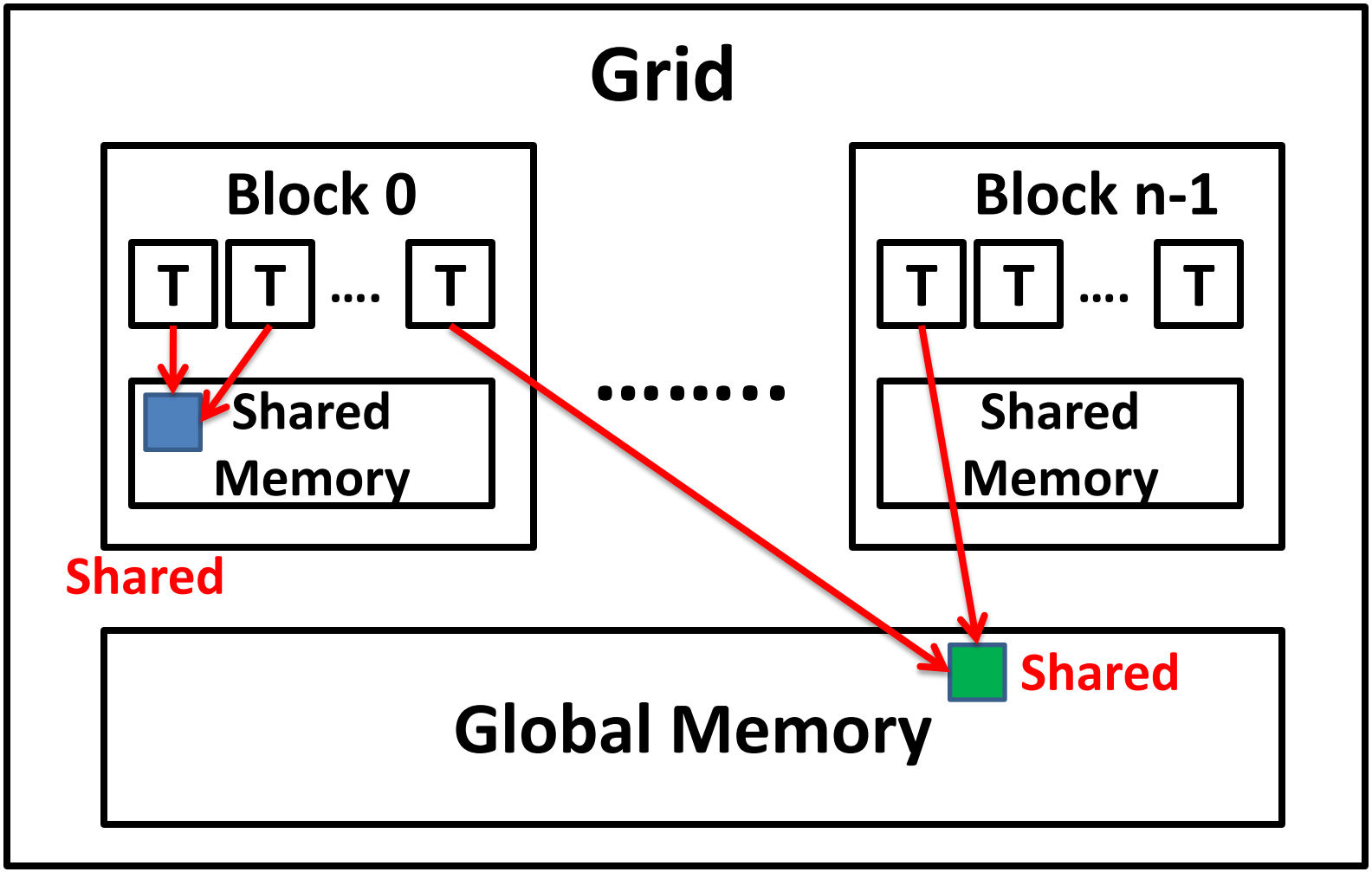
CPU program

```
void inc_cpu(float* a,
             float b, int N) {
  for (int idx = 0; idx<N; idx++)
    a[idx] = a[idx] + b;  }

void main() {
  .....
  increment_cpu(a, b, N); }
```

CUDA program

```
__global__ void inc_gpu(float* A, float b, int N) {
  int tid = blockIdx.x* blockDim.x+ threadIdx.x;
  if (tid < N)
    A[tid] = A[tid] + b;
}

voidmain() {
  .....
  dim3dimBlock (blocksize);
  dim3dimGrid( ceil( N / (float)blocksize) );
  increment_gpu<<<dimGrid, dimBlock>>>(a, b,
N);
}
```

GPU Computation Model

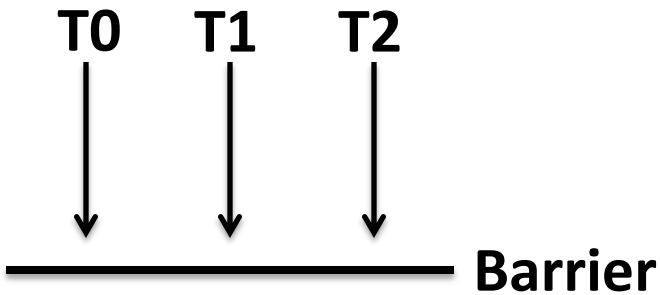


Why is GPU Programming Error-Prone?

- Multi-threaded and shared memory
- Thread synchronization.
  - Barrier
  - Atomic operations
- Safety properties:
  - Races
  - Assertions **Our focus!**

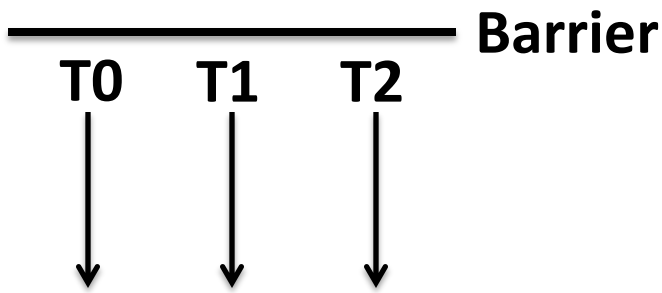
Why is GPU Programming Error-Prone?

- Multi-threaded and shared memory
- Thread synchronization.
  - Barrier
  - Atomic operations
- Safety properties:
  - Races
  - Assertions **Our focus!**



Why is GPU Programming Error-Prone?

- Multi-threaded and shared memory
- Thread synchronization.
  - Barrier
  - Atomic operations
- Safety properties:
  - Races
  - Assertions **Our focus!**



Why is GPU Programming Error-Prone?

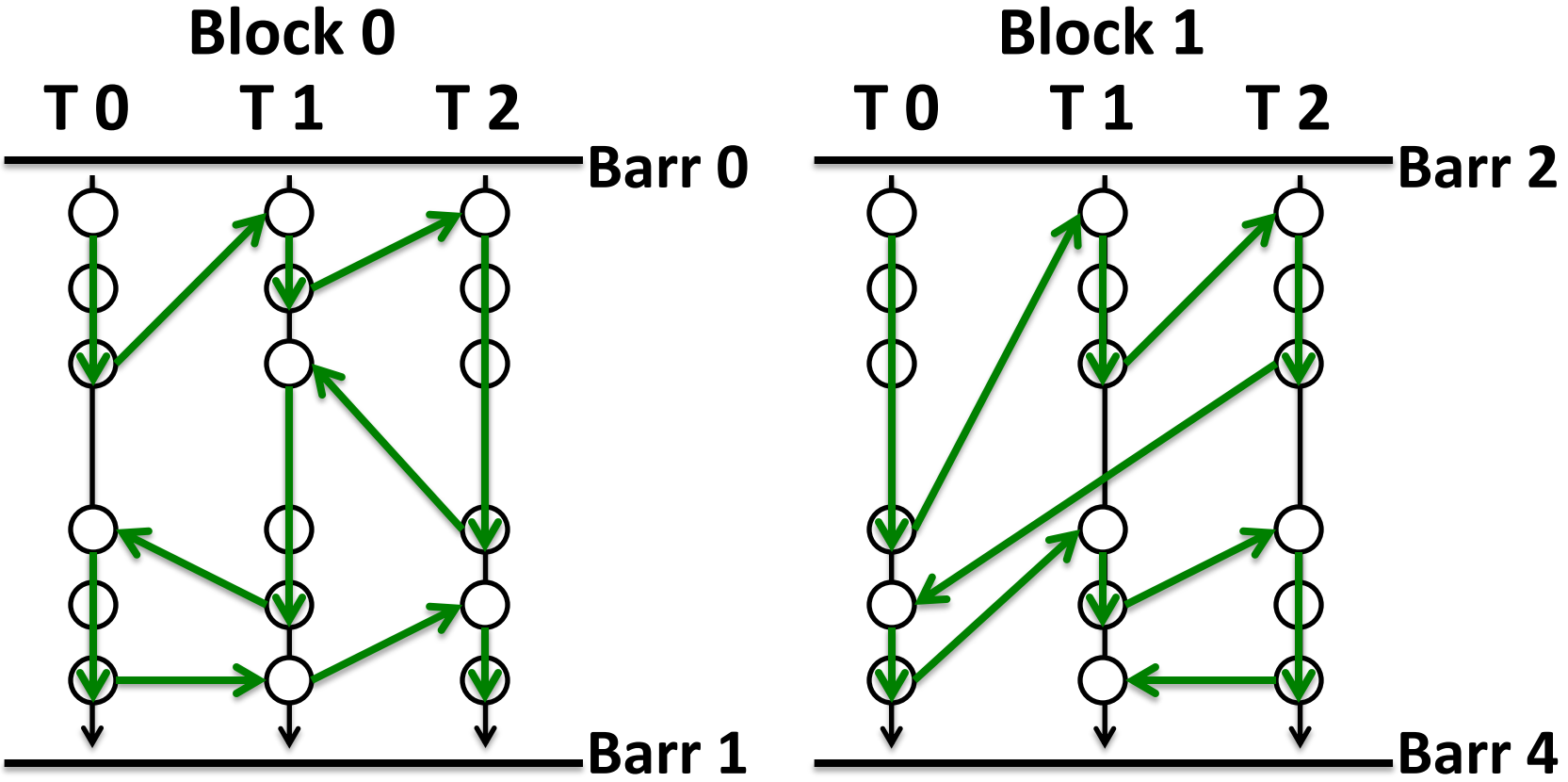
- Multi-threaded and shared memory
- Thread synchronization.
  - Barrier
  - Atomic operations
- Safety properties:
  - Races
  - Assertions **Our focus!**

atomic op. { **read v sh[i]** **write sh[i] v**

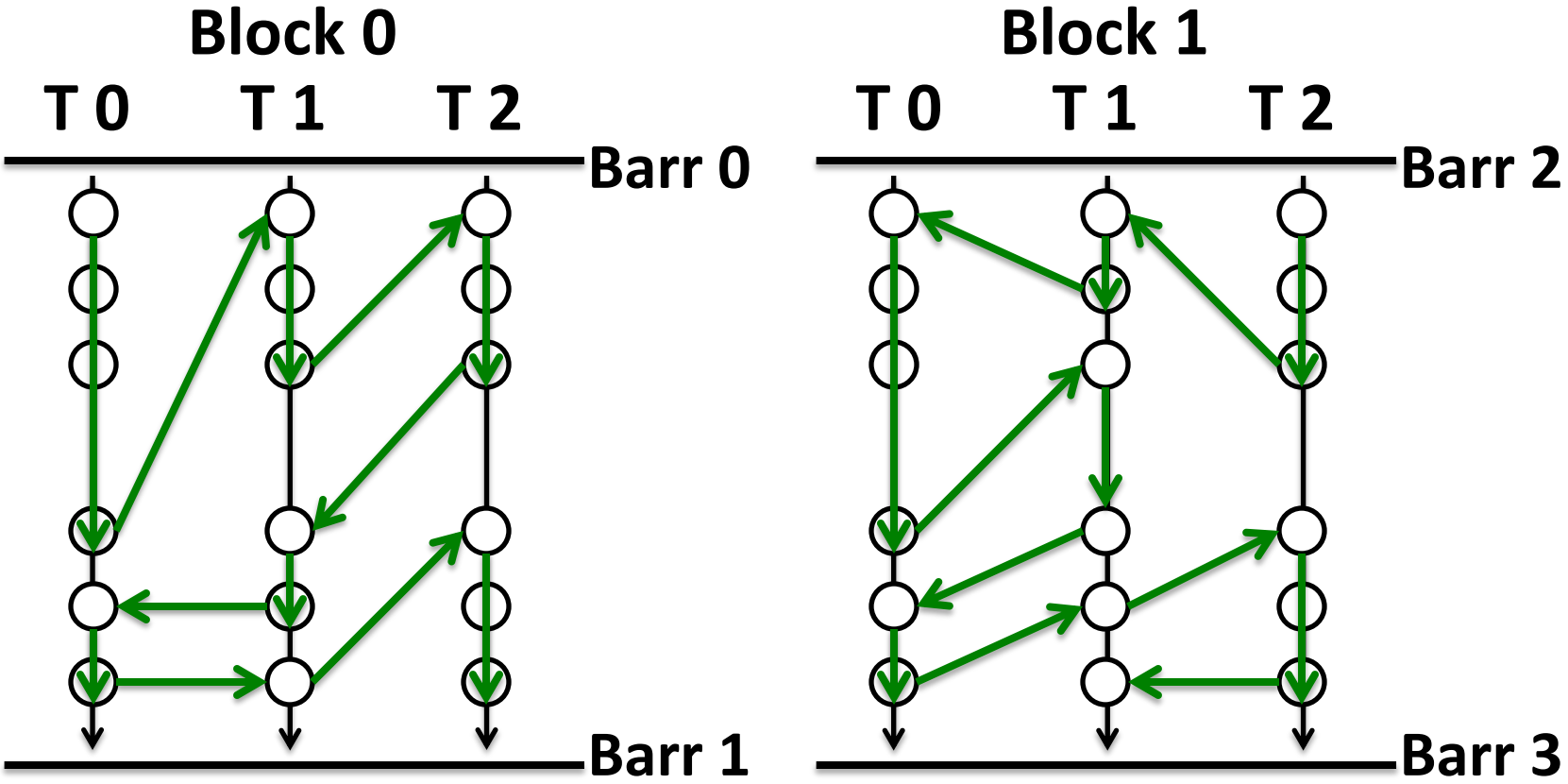
Why is GPU Programming Error-Prone?

- Multi-threaded and shared memory
- Thread synchronization.
  - Barrier
  - Atomic operations
- Safety properties:
  - Races
  - Assertions **Our focus!**

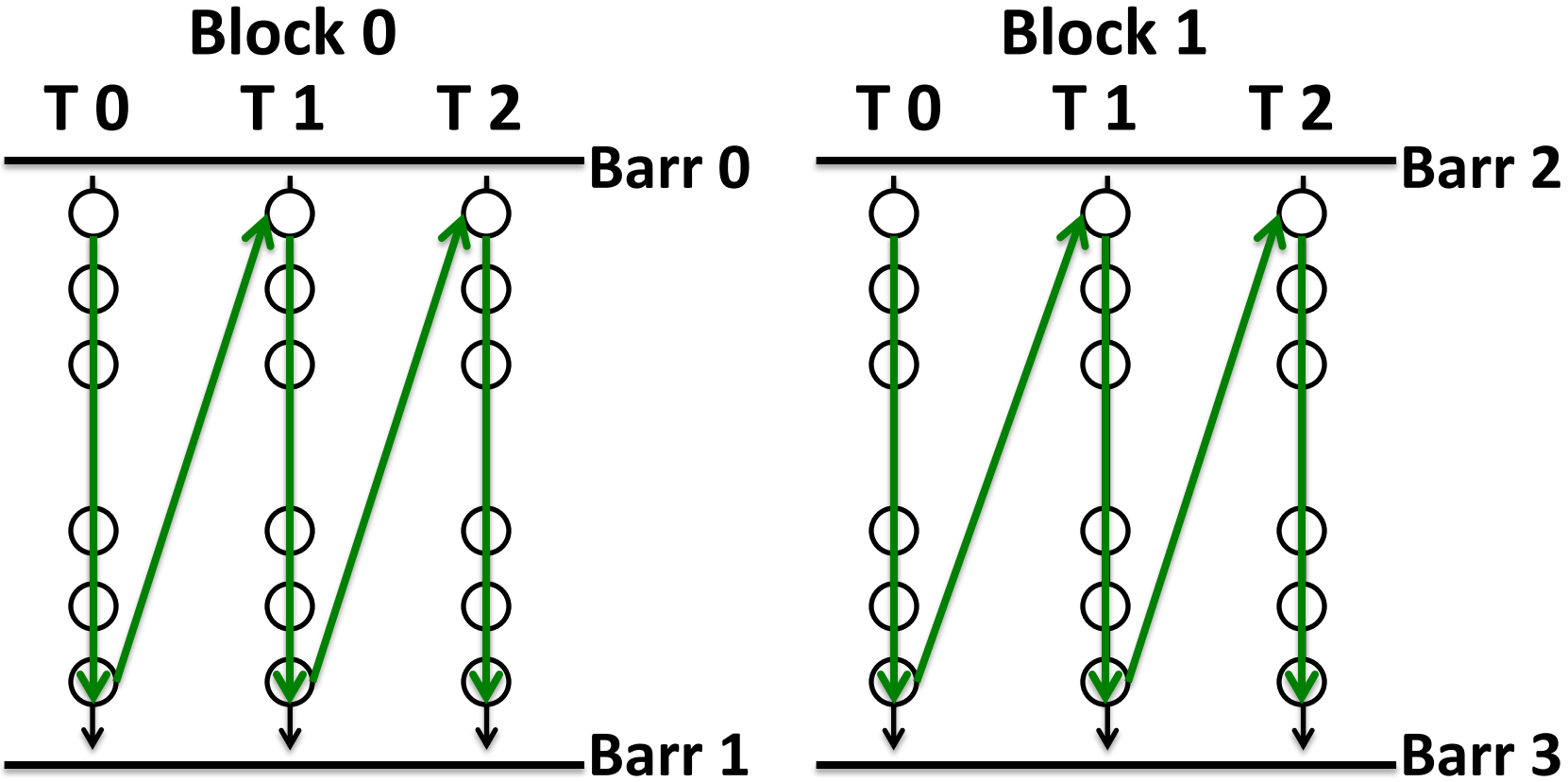
Schedules Are Equivalent in Race-free programs ("DRF theorems")



Schedules Are Equivalent in Race-free programs ("DRF theorems")

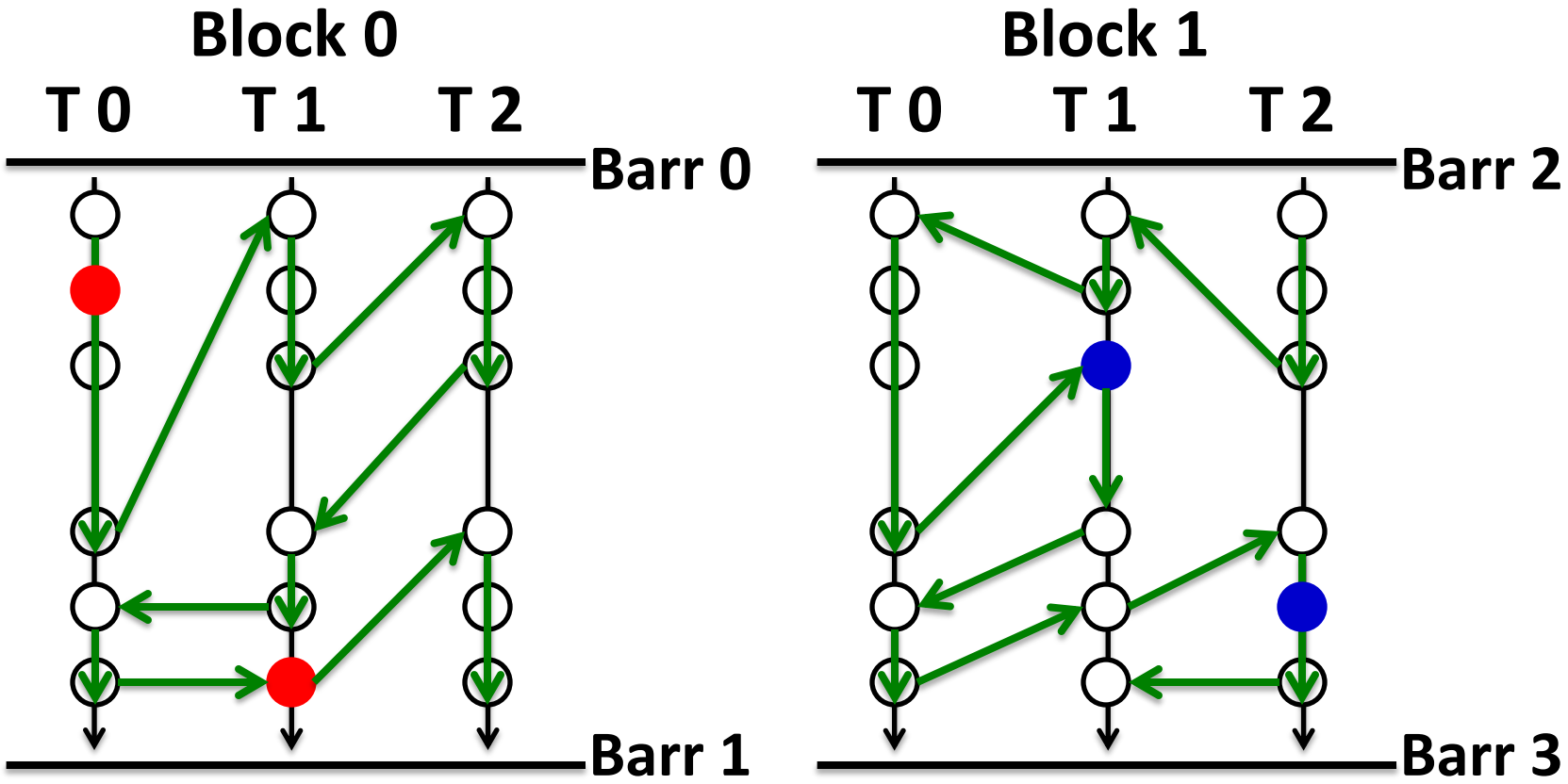


Schedules Are Equivalent in Race-free programs ("DRF theorems")



Sequential Scheduling [Attiya, 1994]

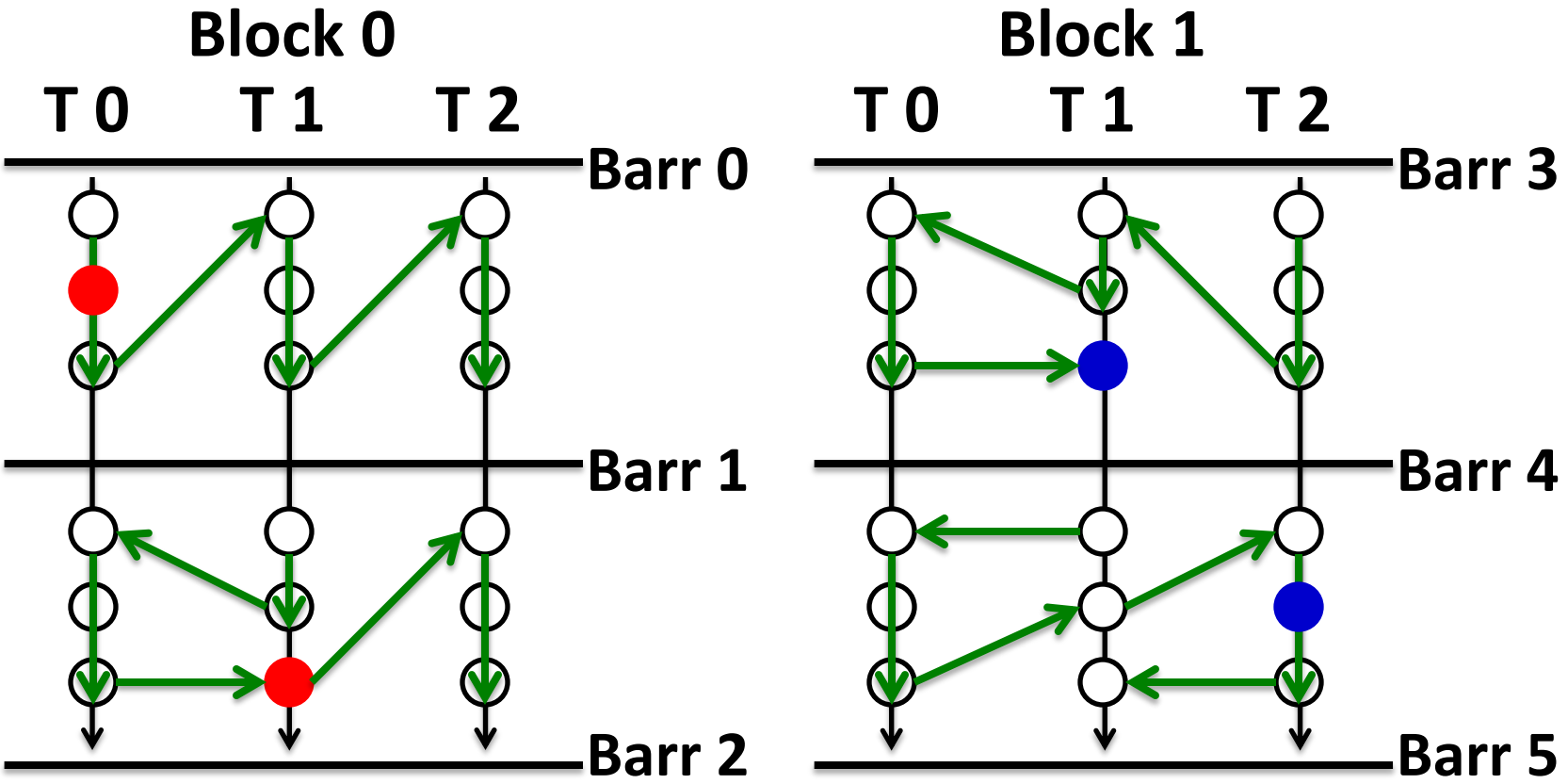
Race Detection by Single Schedule



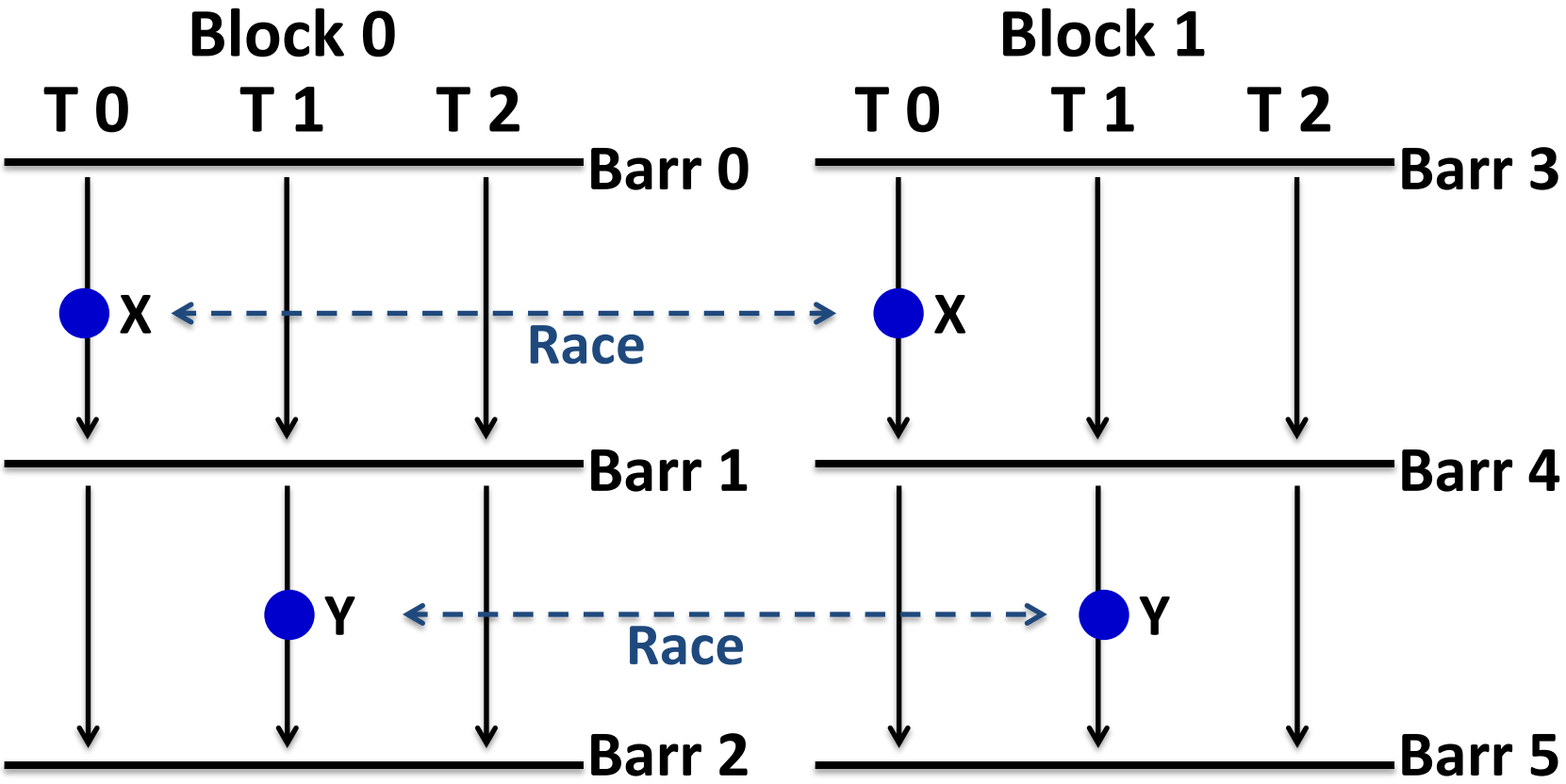
Explore ONE schedule is sufficient to detect races if there is any.

[Adve, 1991] [Li, PPOPP'12]

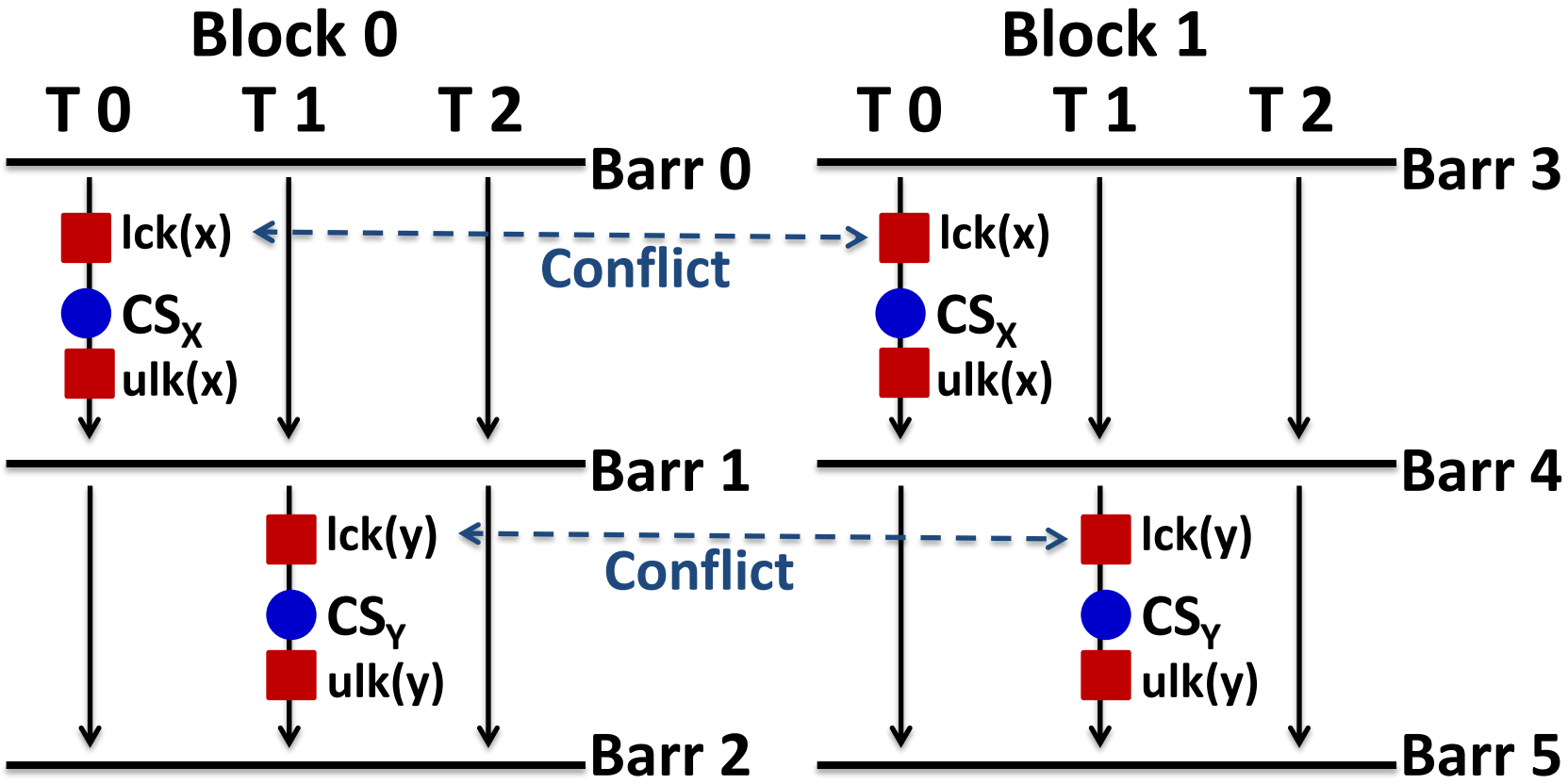
Race Pruning by Introducing Barriers



Barriers don’t prevent Inter-block Race



Needs of Critical Sections

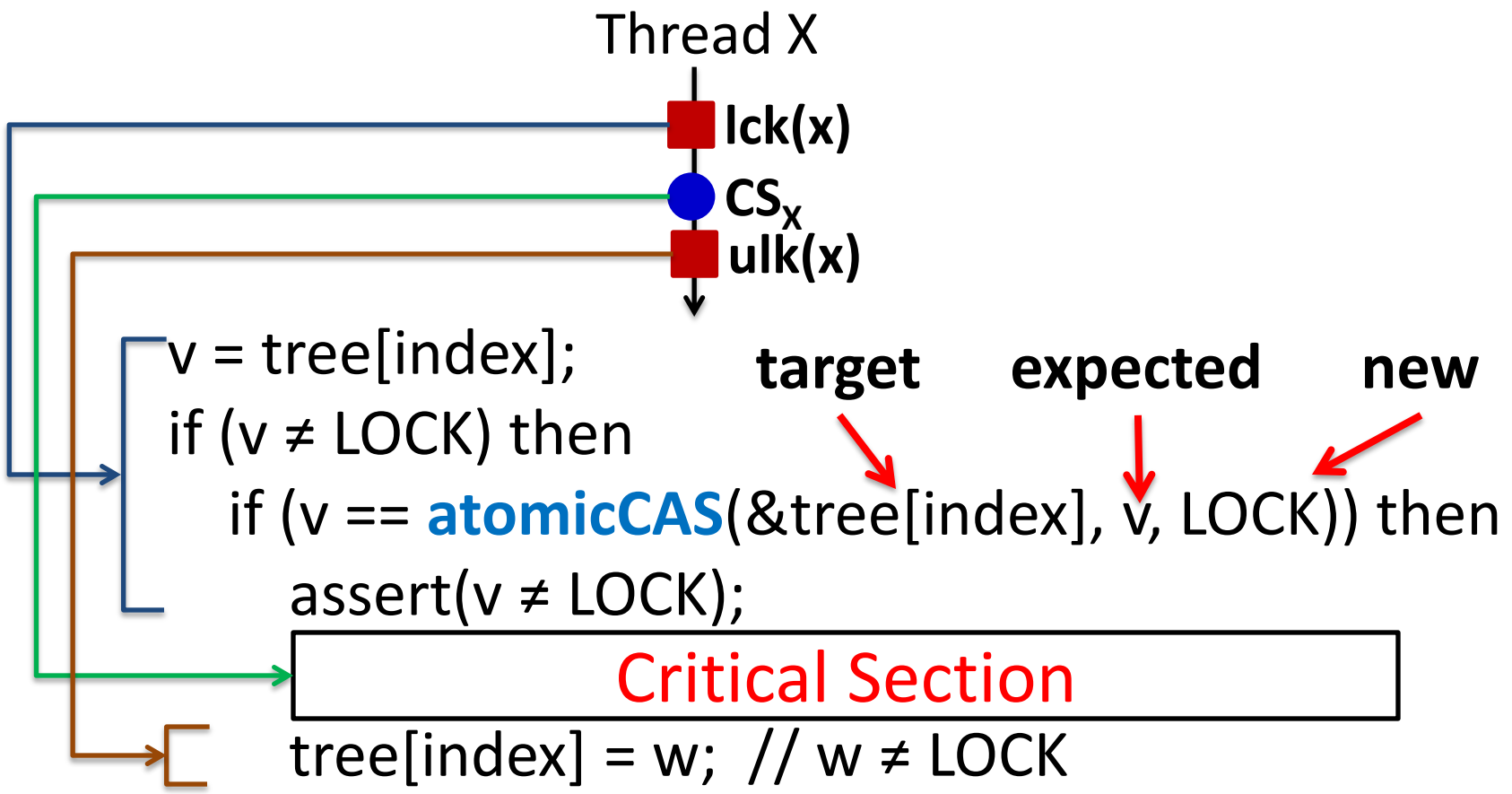


Atomics Based Synchronization Example


```
v = tree[index];
if (v ≠ LOCK) then
    if (v == atomicCAS(&tree[index], v, LOCK)) then
        assert(v ≠ LOCK);
        Critical Section
        tree[index] = w;  // w ≠ LOCK
```

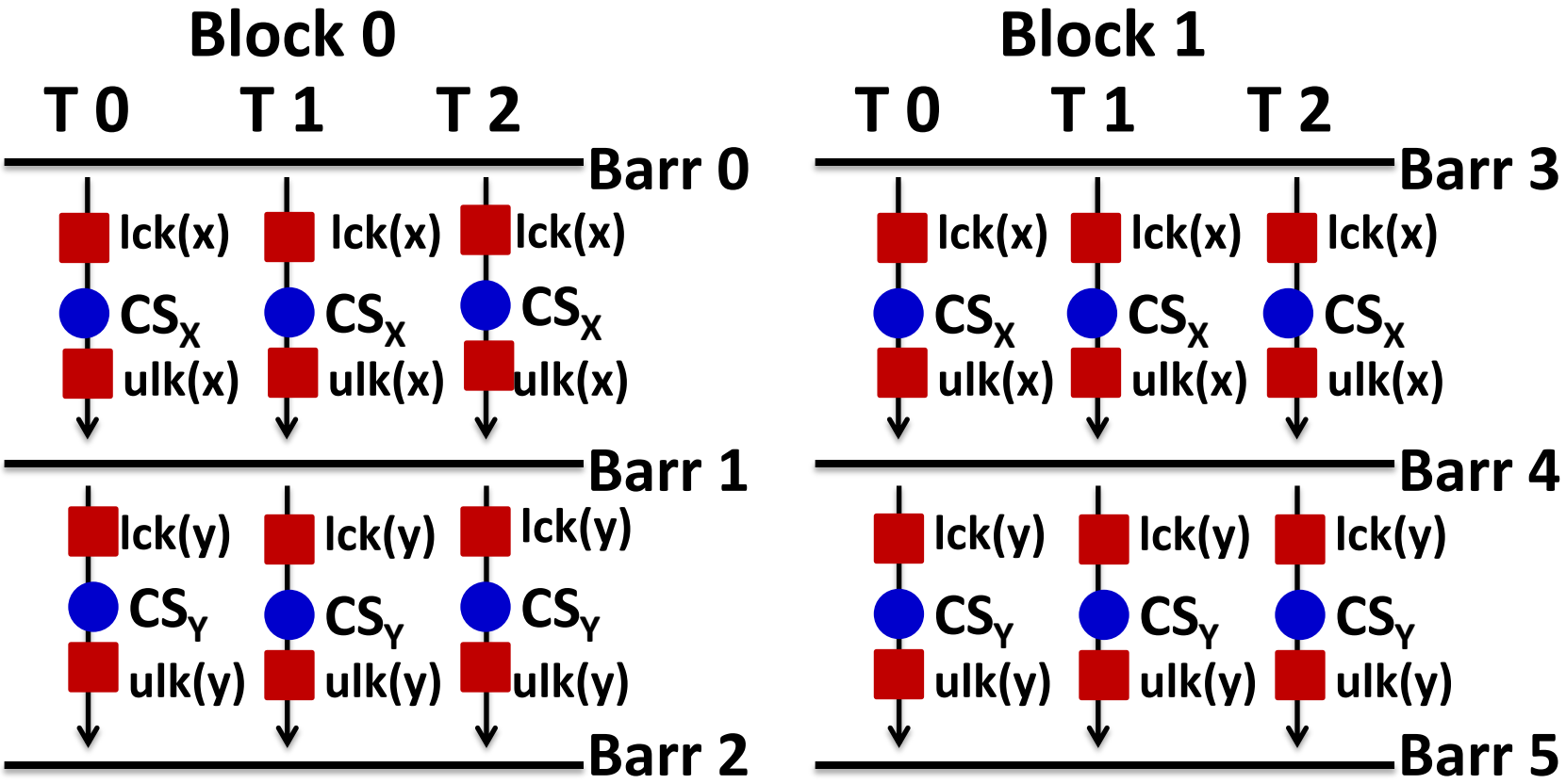
[Burtscher, 2011]

Atomics Based Synchronization Example



[Burtscher, 2011]

Number of Conflicts Could be Many!!



Schedules are not equivalent!!

Schedule Exploration is Needed!

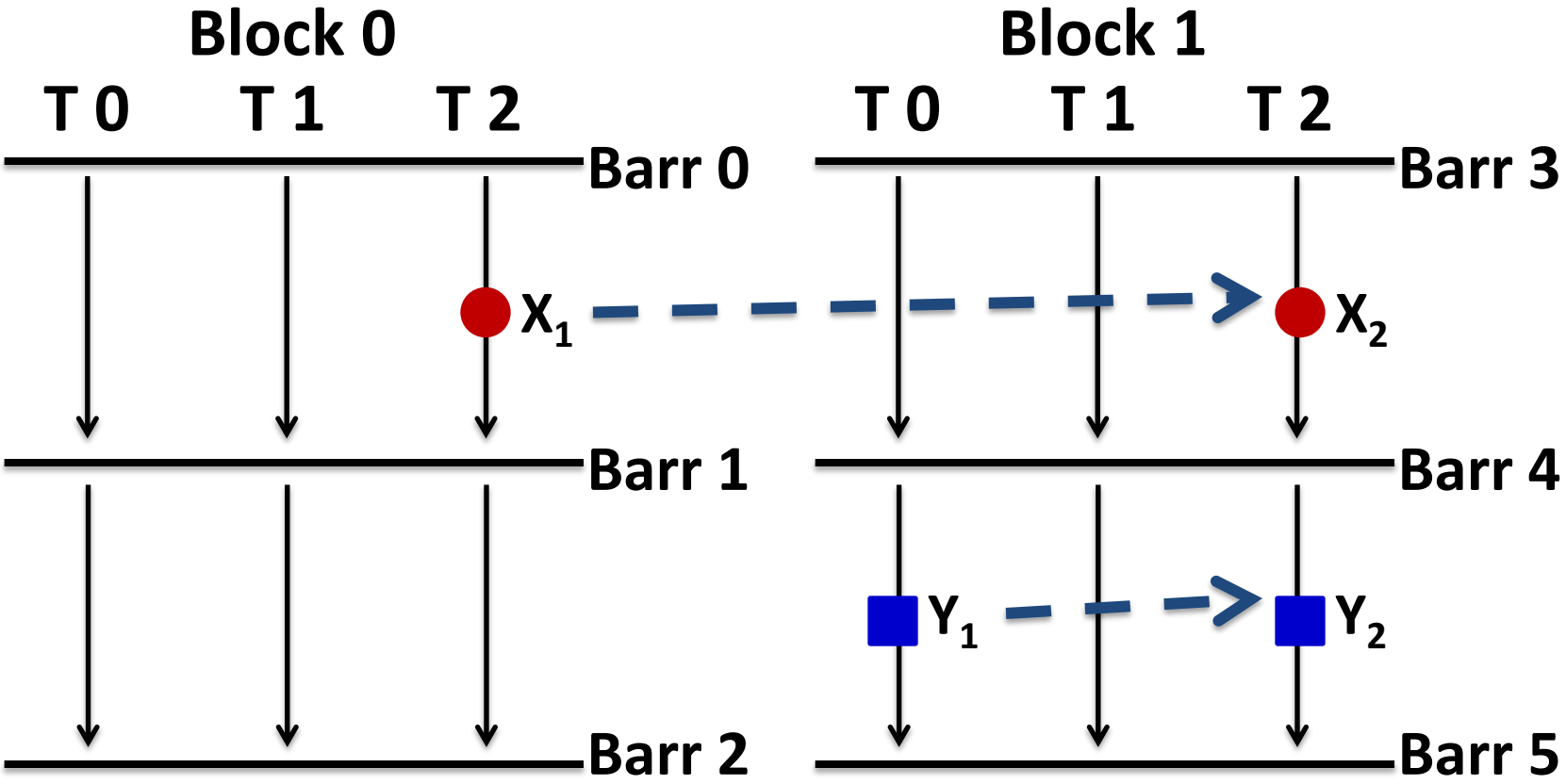
- Previous work
  - Barrier based synchronization
    - Explore one schedule.
- Our work
  - **Barrier+atomic** based synchronization
    - Atomic operations introduce conflicts
    - Explore multiple schedules.
    - Need a good scheduling strategy.

Our Contributions

- **C**onflict-directed **D**elay-bounding (CD) scheduling strategy.
  - Checks safety properties with synchronizations using barriers+atomic.
- Operational semantics of CD scheduling.

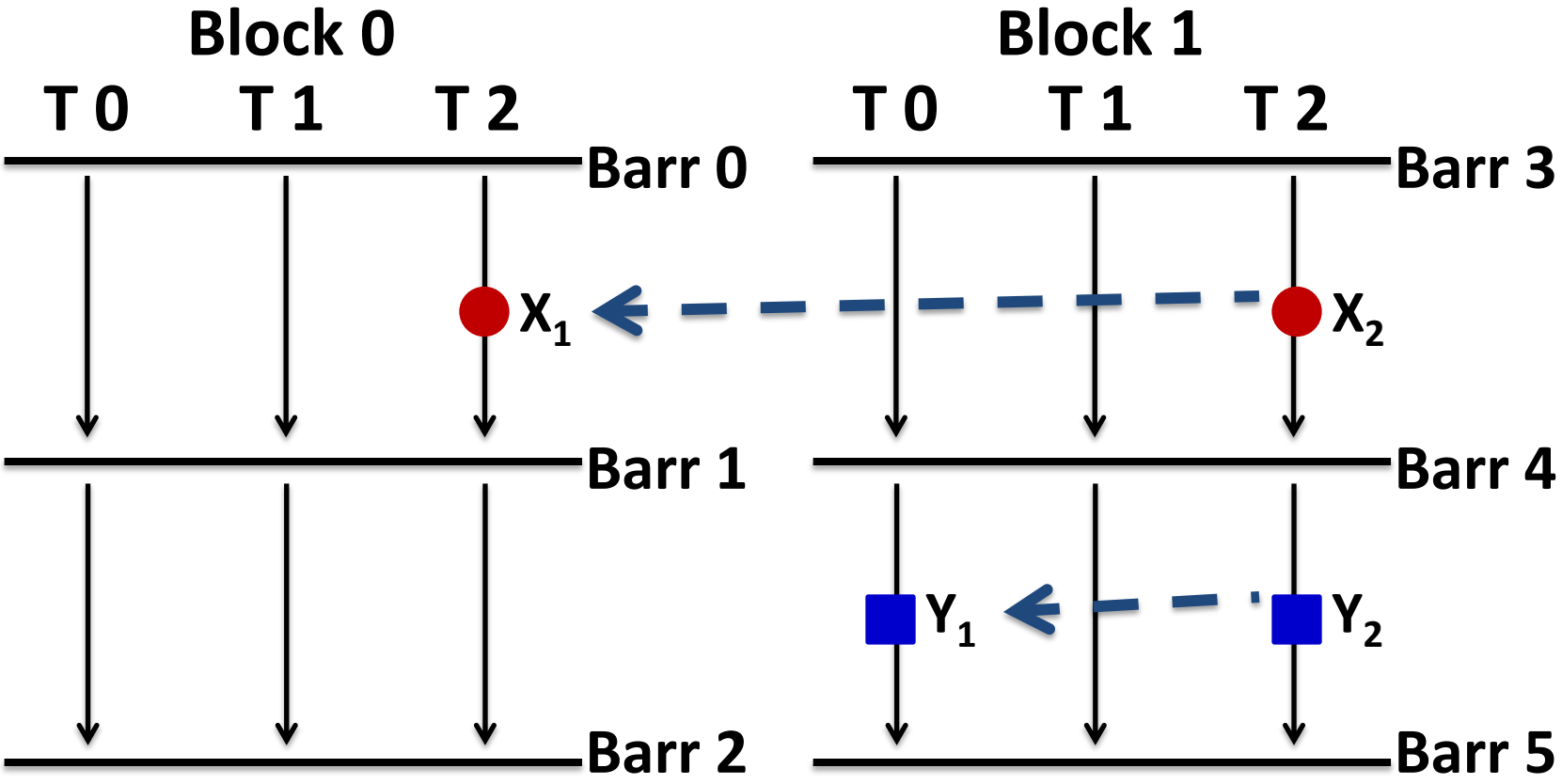
Without a good search strategy →

Intuitions of CD Scheduling



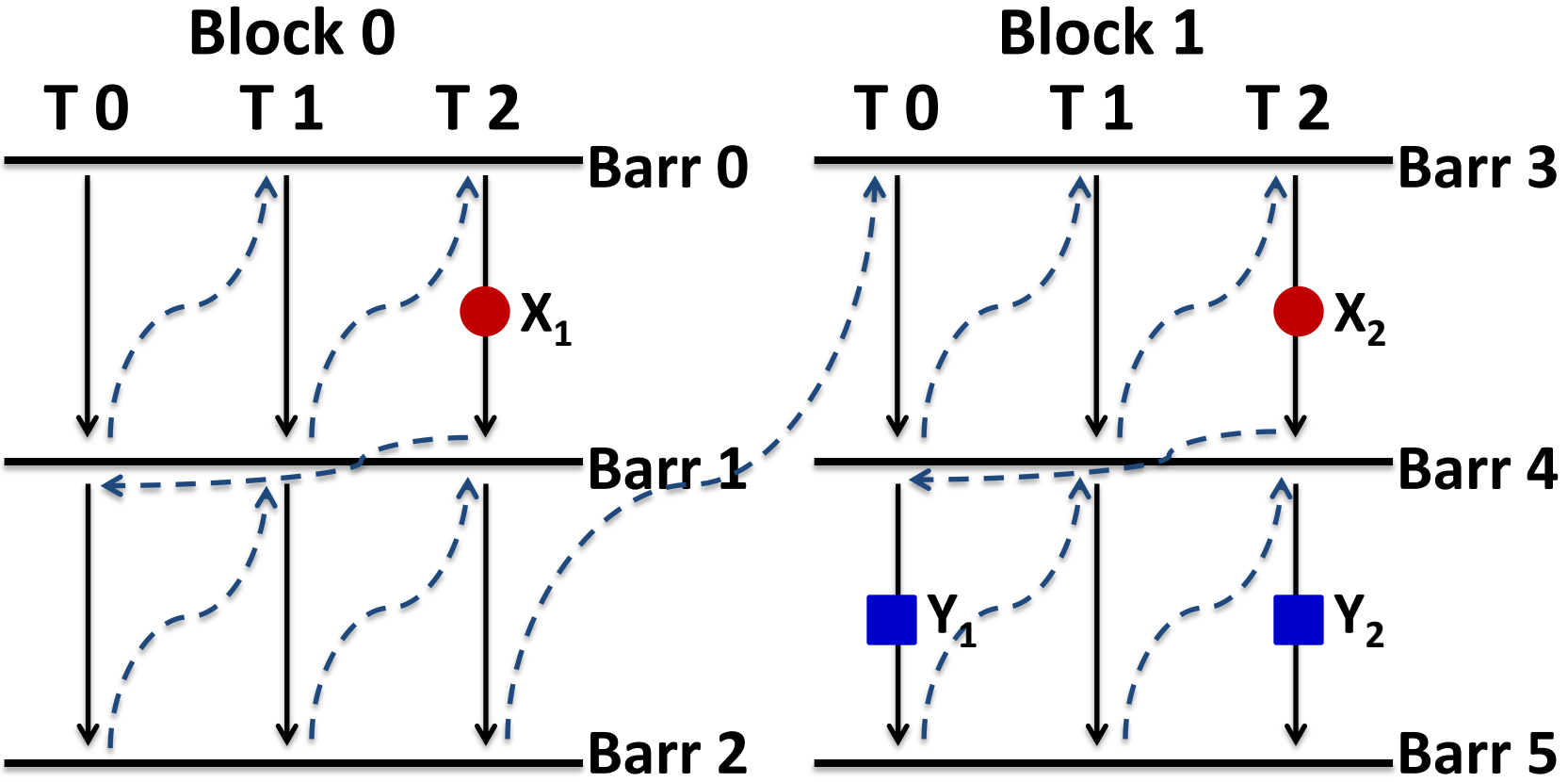
By some schedule, we visit $X_1 \rightarrow X_2$ and $Y_1 \rightarrow Y_2$.

Intuitions of CD Scheduling



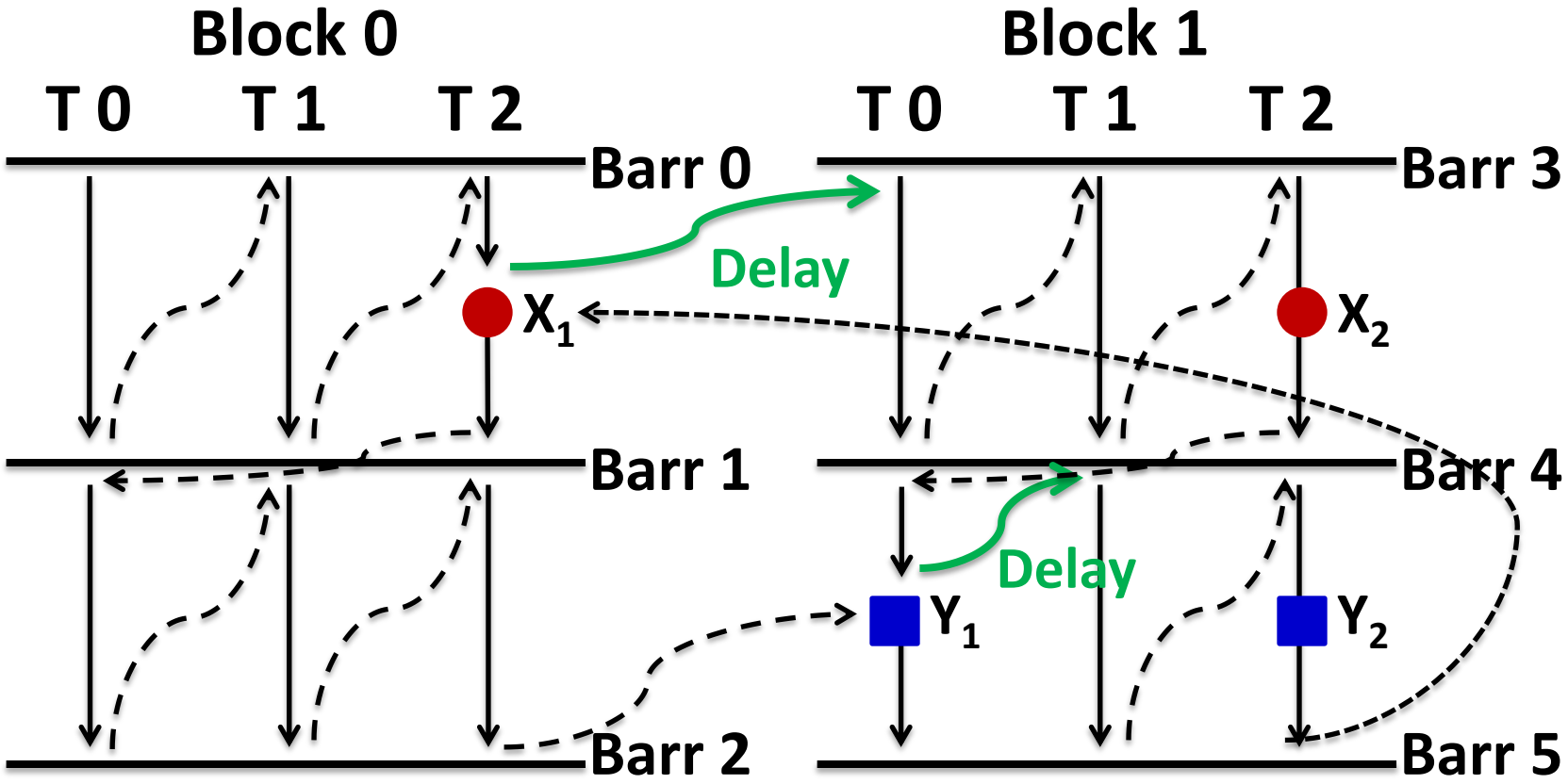
Find another schedule that we visit $X_2 \rightarrow X_1$ and $Y_2 \rightarrow Y_1$.

Background: Sequential Scheduling



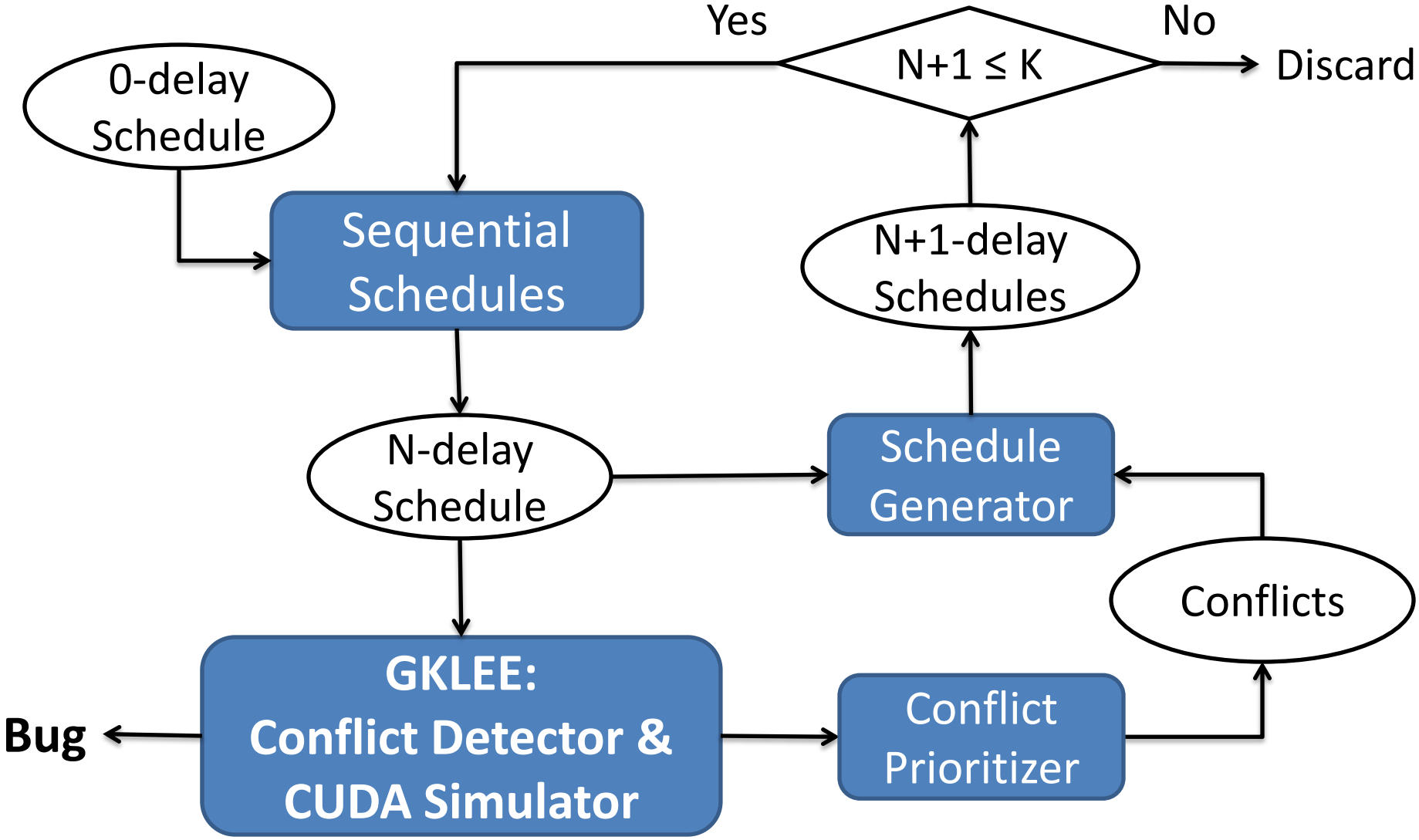
Two Conflicts: $X_1 \rightarrow X_2$ and $Y_1 \rightarrow Y_2$.

CD Scheduling

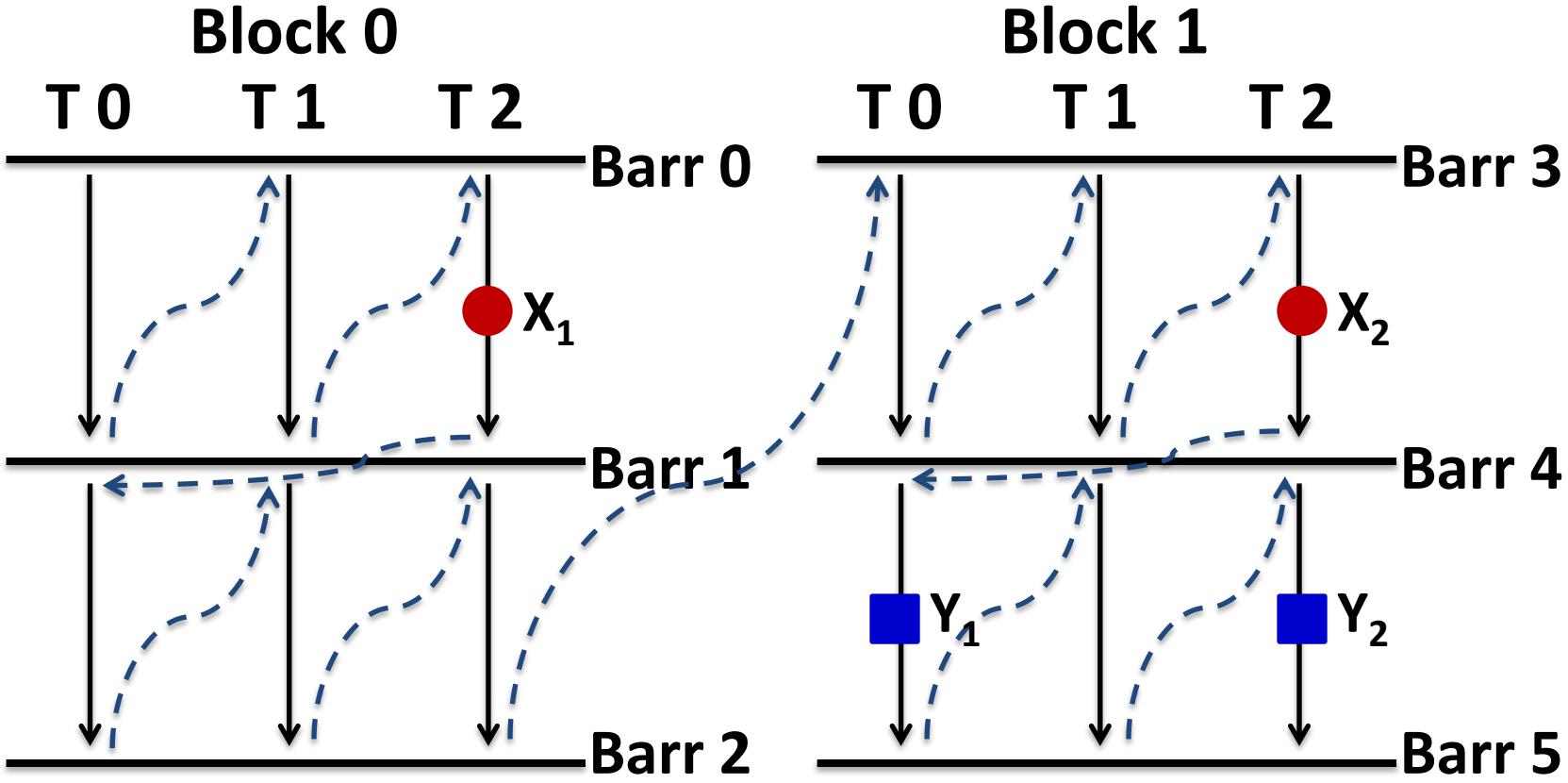


CD Schedule: $X_2 \rightarrow X_1$ and $Y_2 \rightarrow Y_1$.

High-Level View of CD Scheduling

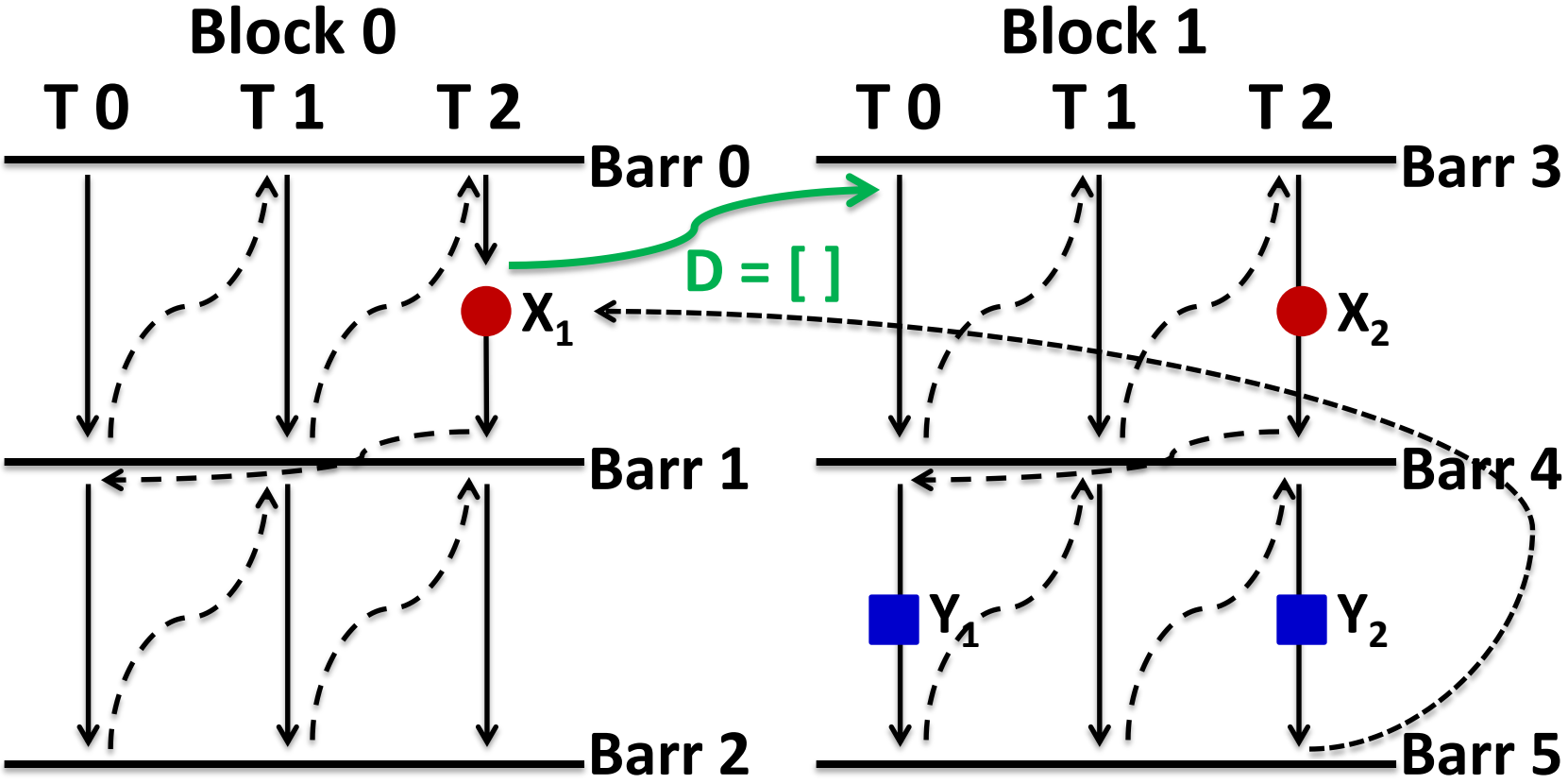


0-delay Schedule: D = [ ]



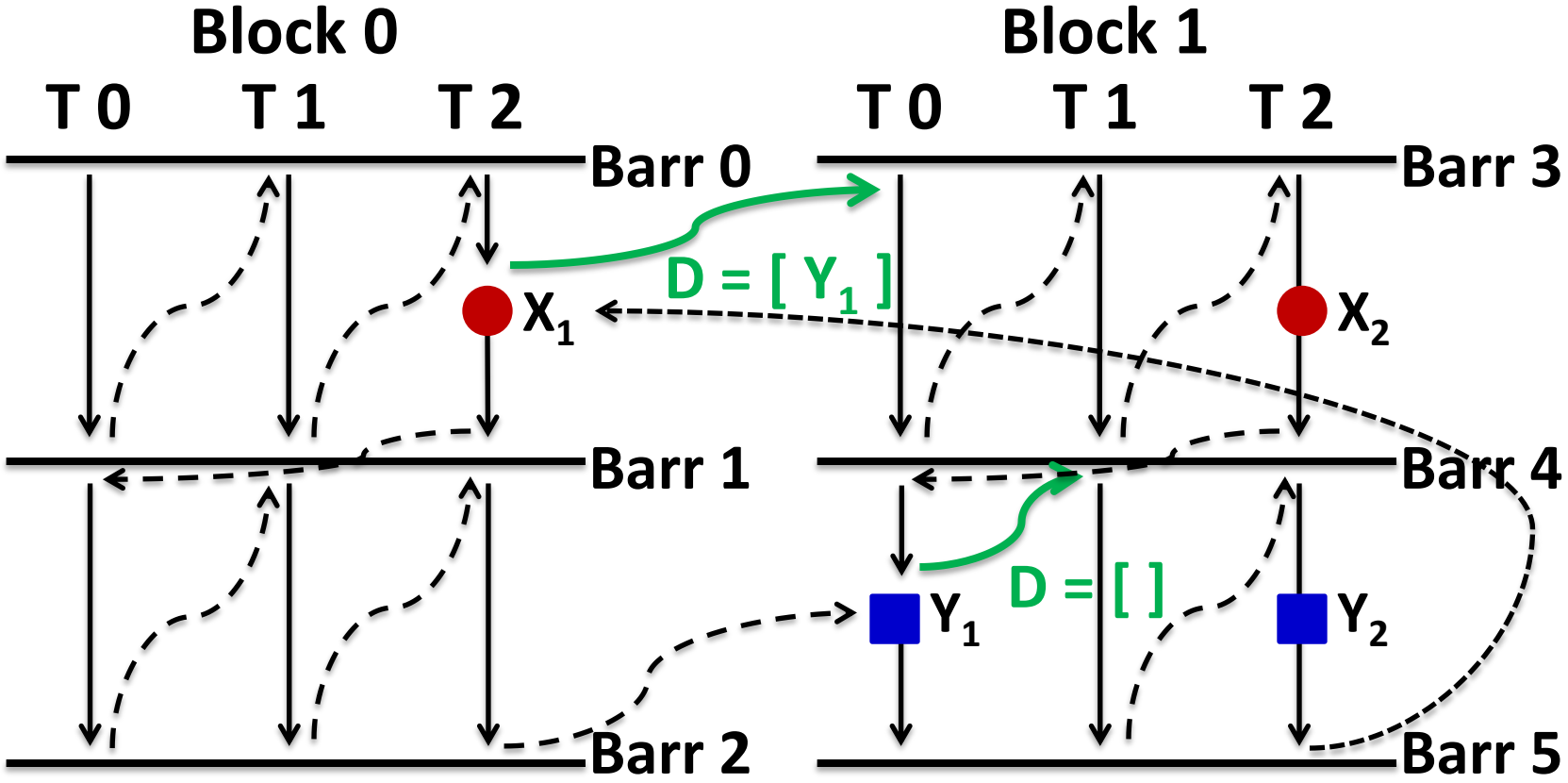
Two Conflicts: $X_1 \rightarrow X_2$ and $Y_1 \rightarrow Y_2$.

1-delay Schedule: D = [X_1]

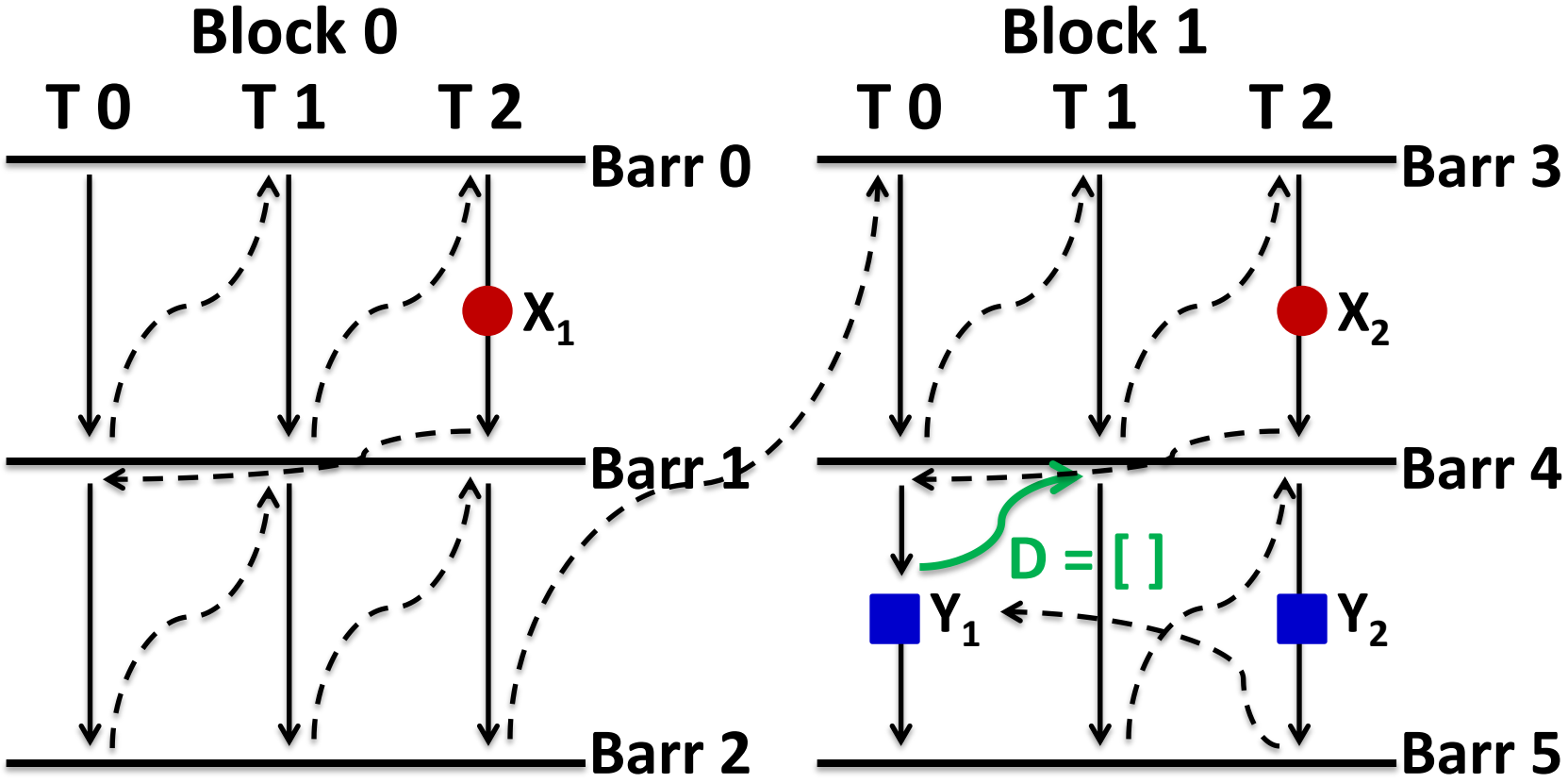


A conflict is detected after D = [ ]: $Y_1 \rightarrow Y_2$.

2-delay Schedule: D = [X_1, Y_1]



1-delay Schedule: D = [Y_1]



No conflict is detected after D = [ ]

Operational Semantics of CD

- Opsem of Scheduling given in our paper
  - Different scheduling options (e.g. which thread to run next) captured using uninterpreted functions
- Limitation of current CD approach:
  - Does not address unfair CUDA runtimes [Habermeir, Knapp, ESOP'13]
  - Will be addressed in our future work

Example: Buggy N-body Simulation

- We **planted** the following bug in N-body code [Burtscher, GCG'11]

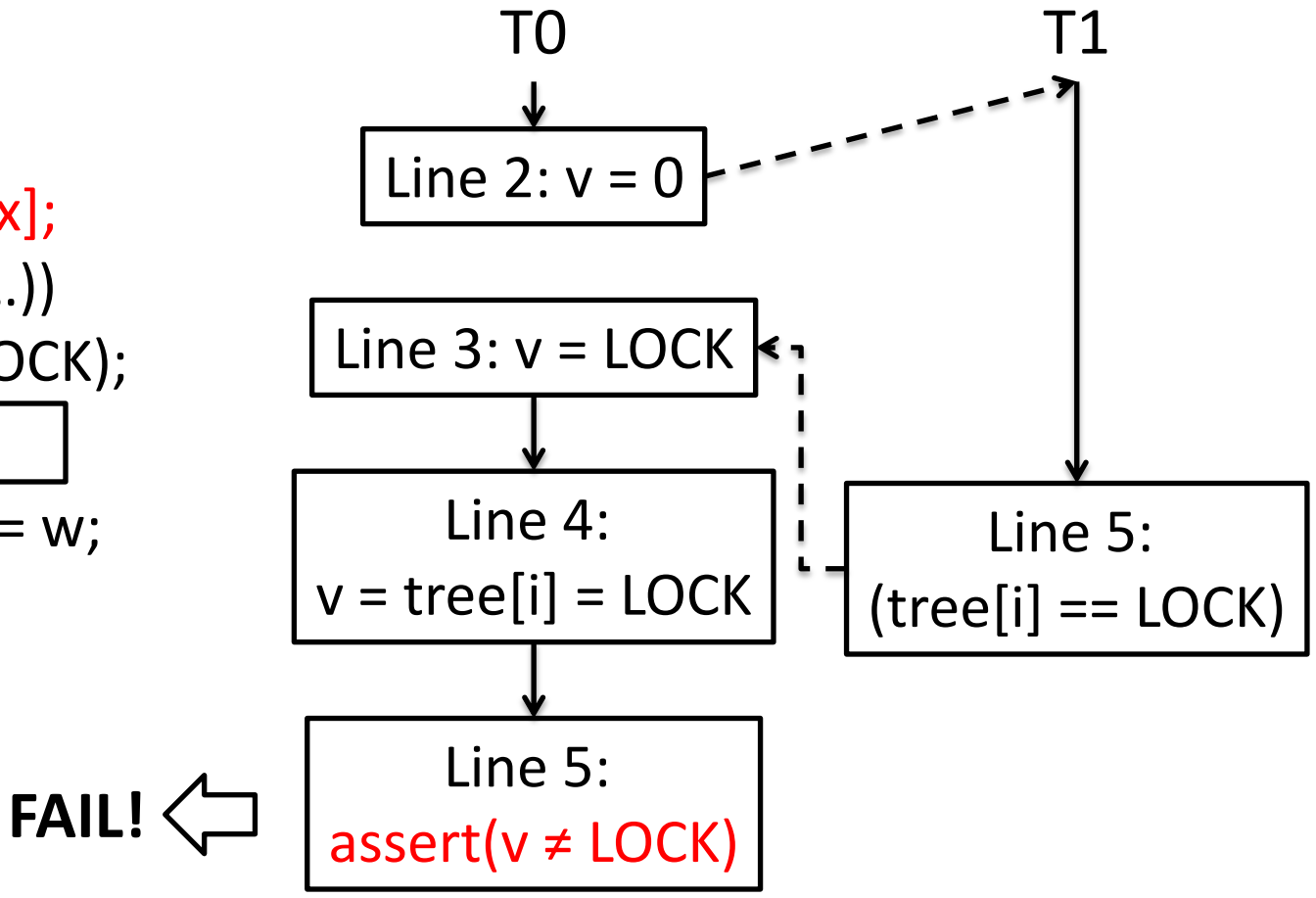
```
1.  v = tree[index];
    (a long section of code here…)
2.  if (v ≠ LOCK) then
3.      v = tree[index];
4.      if (v == atomicCAS(&tree[index], v, LOCK)) then
5.          assert(v ≠ LOCK);
                Critical Section
6.          tree[index] = w;  // w ≠ LOCK
```

Example: Buggy N-body Simulation

1. v = tree[i];
2. if (v ≠ LOCK)
3. v = tree[index];
4. if (v == CAS(…))
5. assert(v ≠ LOCK);

CS

6. tree[index] = w;



Experiments: Bug-Free Benchmarks

Benchmark	LOC	No Heuristic		Heuristic	
		# schedules	result	# schedules	result
aMin	20	431	Verified	431	Verified
aMinUpdate	35	653	Verified	294	Verified
bintree	75	835	Verified	405	Verified
TSP	130	114	Verified	60	Verified
N-body	260	1195	Verified	336	Verified

- Heuristic: pick “conditional atomic operation” conflicts (atomicCAS)
- Errors are detected using user-provided assertions
- 3 blocks and 1 thread per block, delay-bound is 2
- Running times ranging from 5 to 5000 seconds

Experiments: Buggy Benchmarks

Benchmark	No Heuristic		Heuristic	
	# schedules	result	# schedules	result
aMin	107	Bug caught	107	Bug caught
aMinUpdate	6	Bug caught	4	Bug caught
bintree	14	Bug caught	202	Omission
TSP	4	Bug caught	4	Bug caught
N-body	448	Bug caught	126	Bug caught

Related Work

- Exploring Seq. Schedules under Race-Freedom
  - General Concurrency Arena:
    - Adve and Hill, ’91
    - “DRF theorems” in Java Memory Model studies
  - GPU Arena:
    - Li and Gopalakrishnan, FSE’09
      - Tool : “PUG”
    - Li, Li, Gopalakrishnan, Rajan, Ghosh, PPoPP’12
      - Tool : “GKLEE”
    - Betts, Chong, Donaldson, Qadeer, and Thomson, SPLASH’12
      - Tool : “GPUVerify”

Related Work

- **Scheduling Methods:**
  - DPOR
    - Flanagan and Godefroid, POPL'05 (http://users.soe.ucsc.edu/~cormac/)
  - Sequentialization
    - Lal and Reps, CAV'08
    - Torre, Madhusudan, Parlato, CAV'09
    - Nagafi, Hu, Rakamaric', SPIN'10
  - Delay Bounding: Emmi, Qadeer, Rakamaric', POPL'11
- **This work in comparison with above scheduling methods:**
  - Specializes bounding strategy to exploit warp-level sequential scheduling (GKLEE, PPoPP'12)
  - exploits conflicts (Sen, PLDI'08) to schedule around CUDA Atomics

Summary

- Introduced **C**onflict-directed **D**elay-bounded (CD) scheduling search strategy for handling atomics
- Implemented in GKLEE
  - Finds bugs in realistic benchmarks
- Heuristic for picking relevant conflicts
  - Works well in practice

Future Work

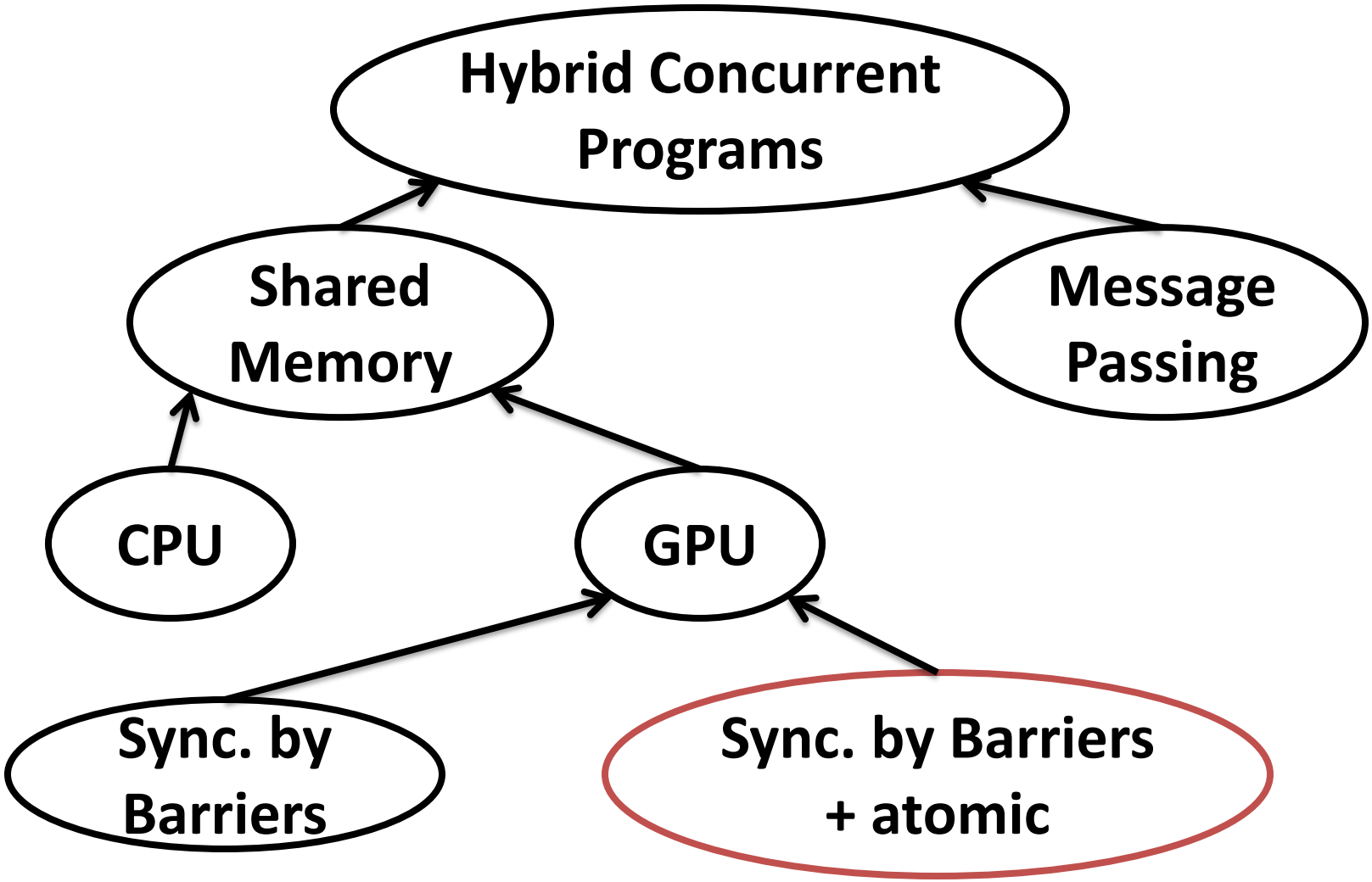
- Extend CD scheduling to other contexts
  - Hybrid programming
  - Recursive calls in GPU kernels
- Address CUDA unfair runtimes
- Include other scheduling strategies such as exploiting thread symmetry

Thanks.

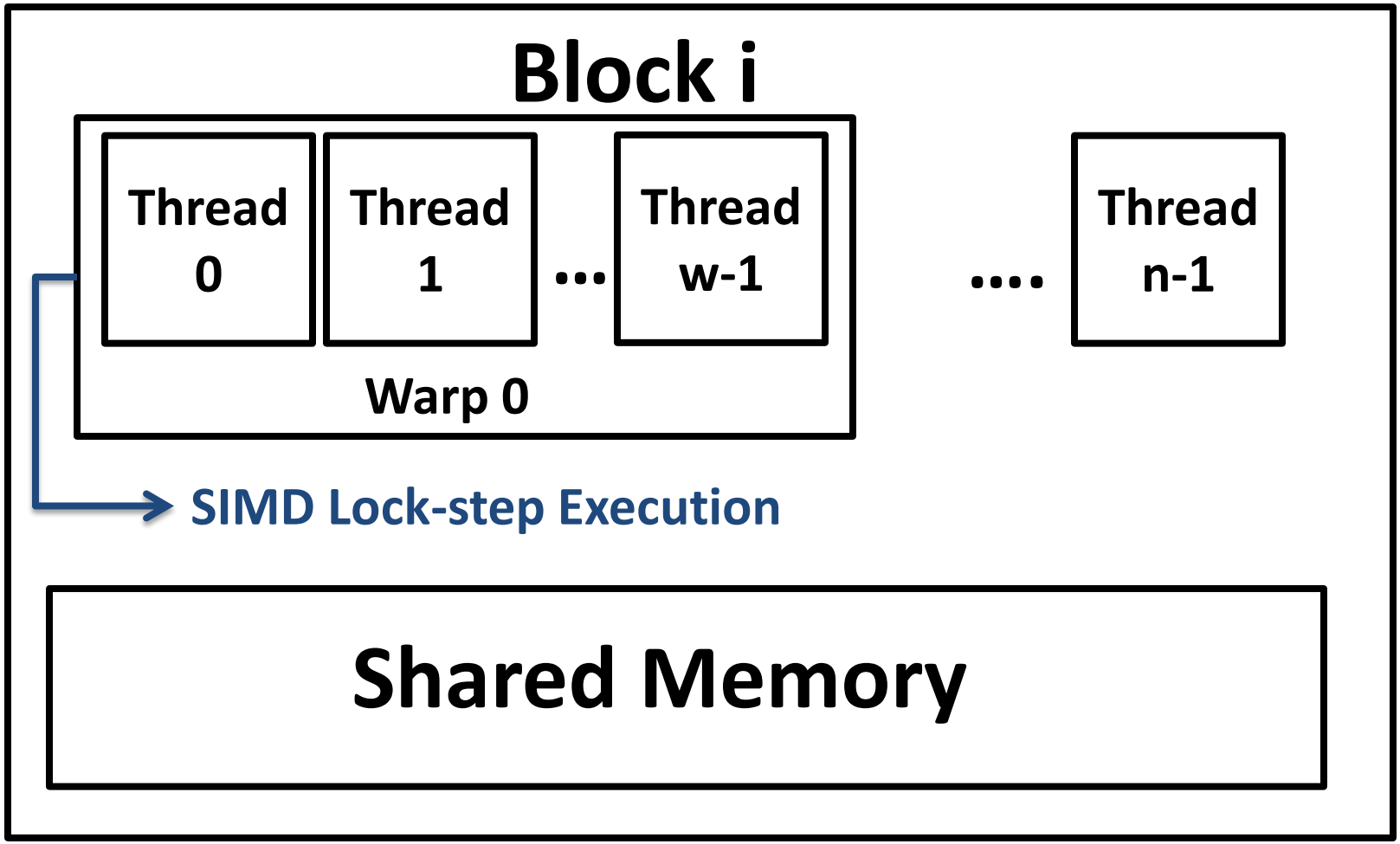
Question?

The following slides are backup slides.

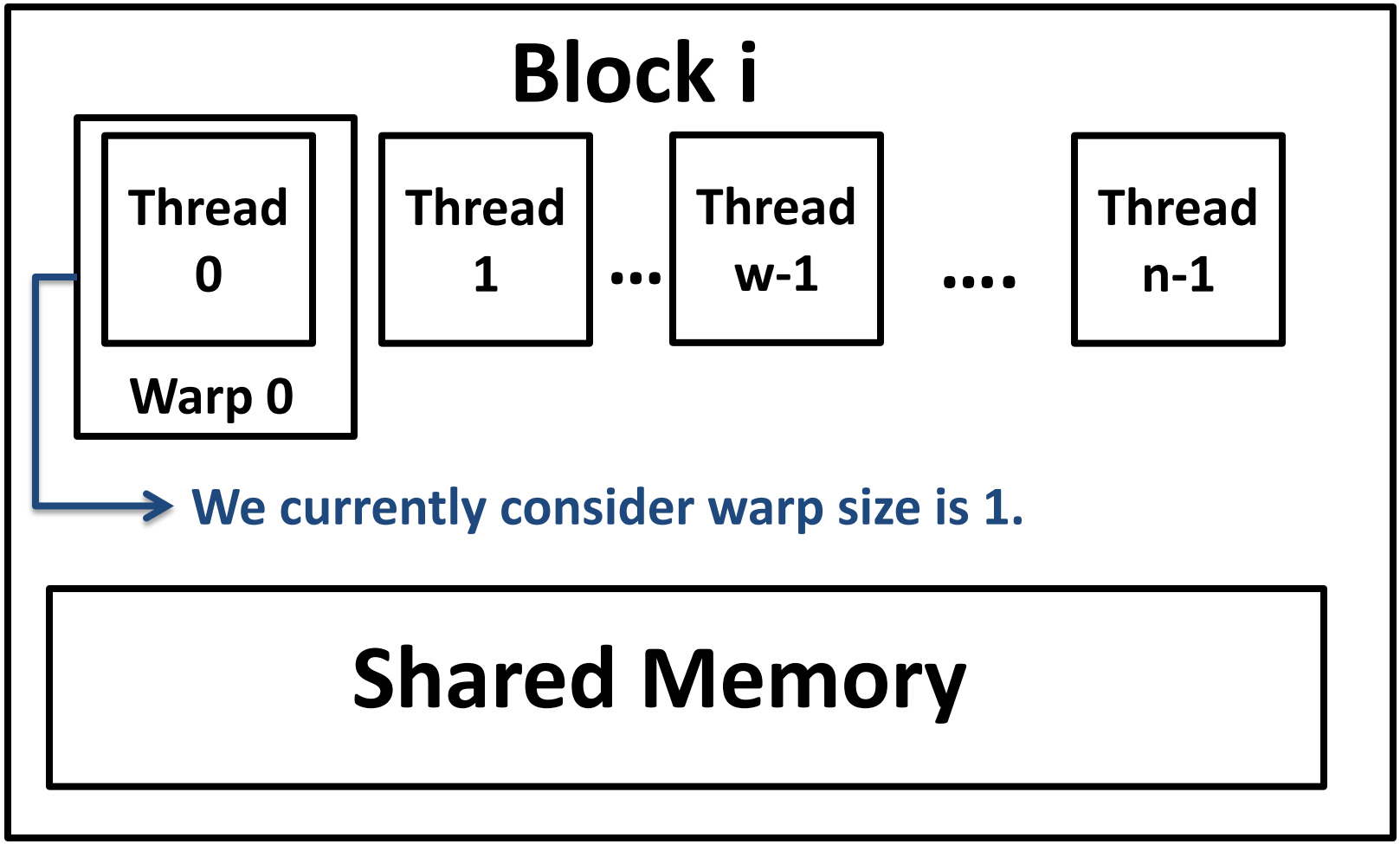
Motivation


Hybrid Concurrent Programs
Shared Memory
Message Passing
CPU
GPU
Sync. by Barriers
Sync. by Barriers + atomic

GPU Computation Model



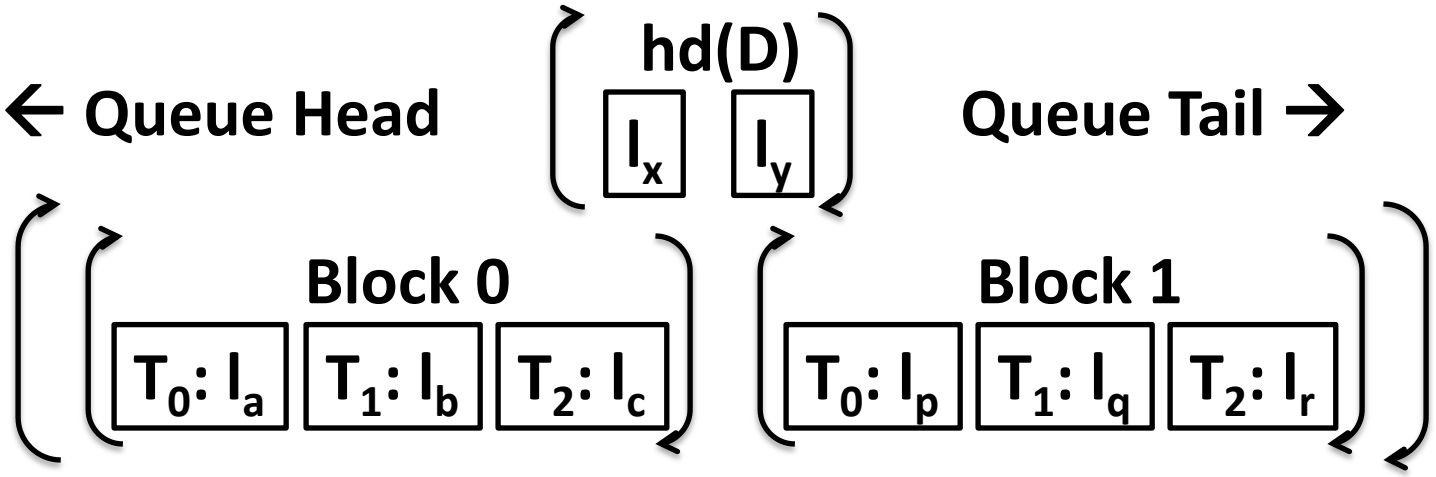
GPU Computation Model



About Handling Warps

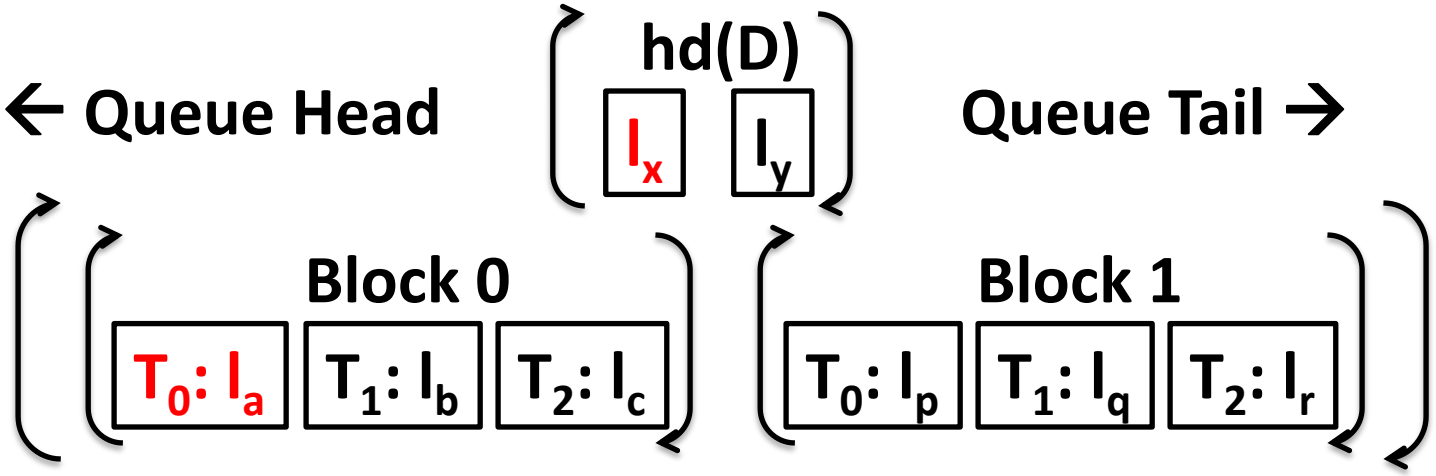
- Our current opsem is based on warp size = 1
- GPU (CUDA) programmers should not assume that warp size is fixed to a certain number
- Thus, assuming warp size = 1 in testing is a heuristic for identifying most races/bugs
- By incorporating the predicated form of CUDA semantics proposed in GPUVerify [Donaldson, 2012], our opsem could also handle warps

Operational Semantics of CD



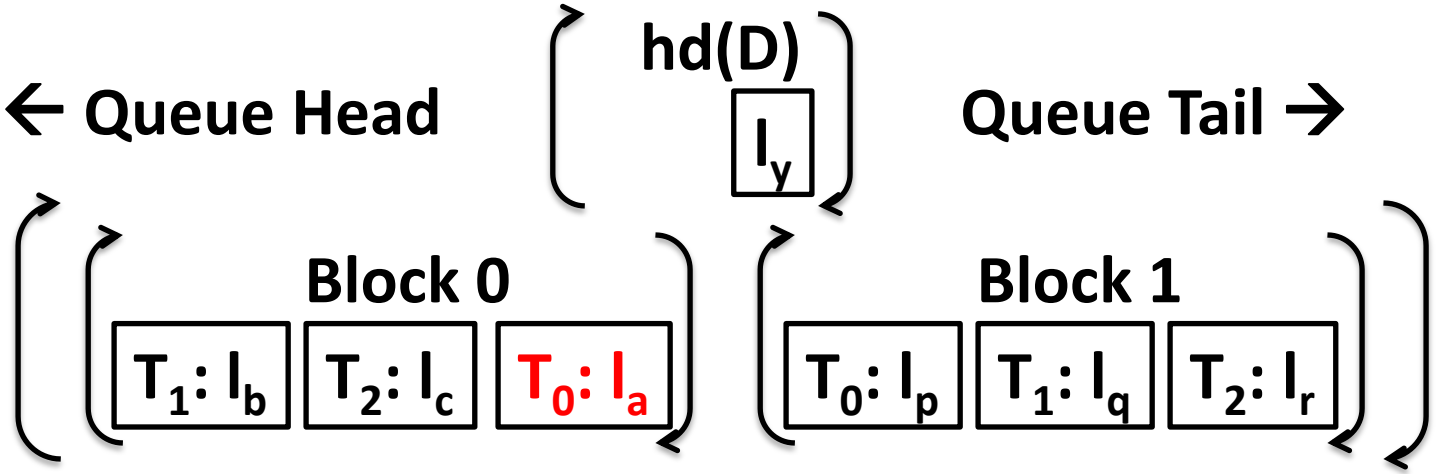
- Blocks and Threads are organized as a queue of queues
- l_a, l_b,… are instructions. "T_0: l_a" denotes that T_0's current instruction is l_a
- hd(D) is the delay set of the current schedule, which is a queue of instructions

Operational Semantics of CD



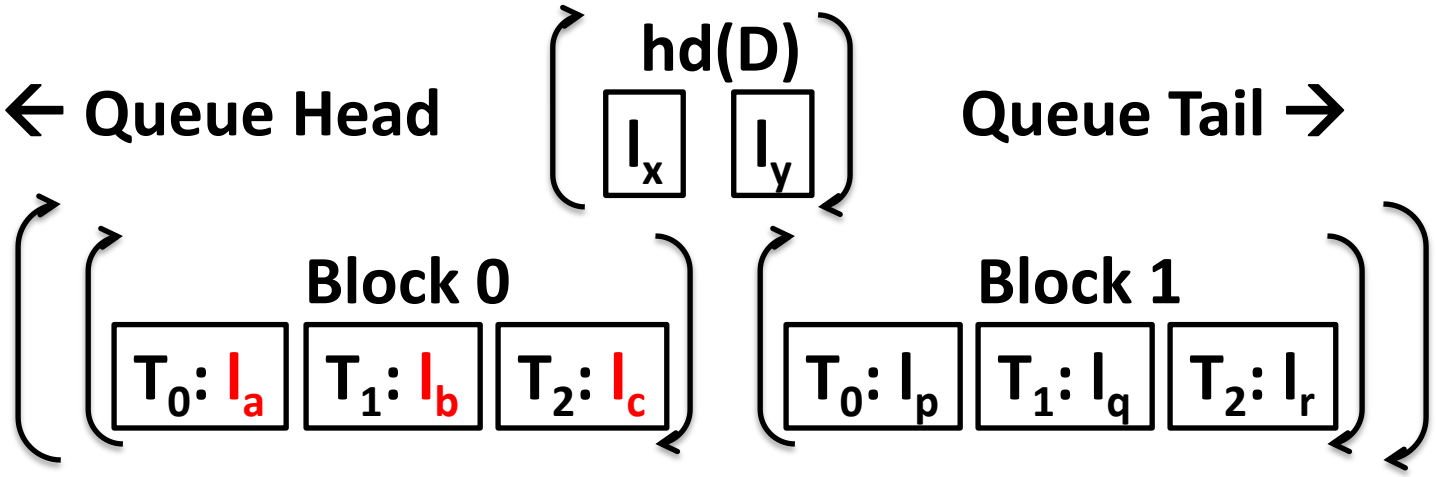
- The 1st thread of the 1st block in the queue is always first considered for scheduling
- Check if l_a is equal to l_x

Operational Semantics of CD



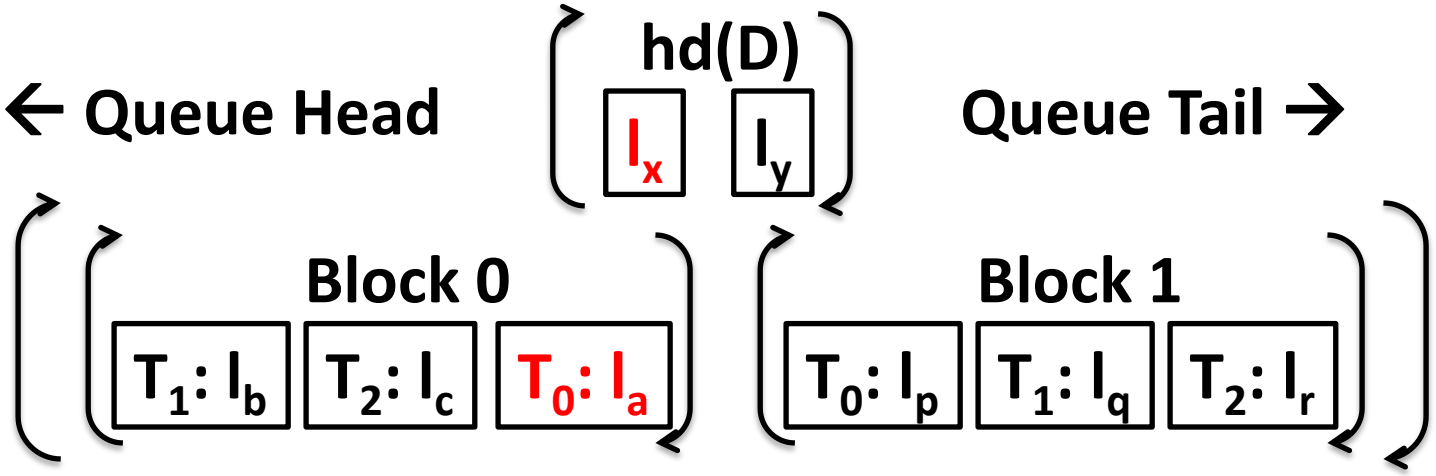
- Precondition: $l_a = l_x$
  - Delay the execution of l_a
- Each instruction in hd(D) is only delayed once

Operational Semantics of CD



- Precondition: $l_a \neq l_x$ and current instructions of all threads in the queue head are barriers, i.e., $l_a = l_b = l_c$= barrier
  - Schedule all threads in the queue head block

Operational Semantics of CD



- Precondition: $l_a \neq l_x$ and there is a thread in the queue head whose current instruction is not a barrier
  - If l_a is not a barrier, execute it
  - Otherwise, the 1st thread of the 1st block (T_0) yields

Related Work: Other GPU Operational Semantics

- Modeling warp execution and divergence.
  - Predicated execution model [Alastair and Qadeer, 2013].
    - A GPU verification tool, GPUVerify, is based on this semantics.
  - Stack-based execution model [Habermaier and Knapp, 2013].
- Our operational semantics models sequential GPU simulation and scheduling strategies.

Comparison between CD Scheduling and Other Strategies

- Vs. Dynamic Partial Order Reduction [Flanagan and Godefroid, 2005]
  - CD scheduling priorities schedule explorations with detected conflicts.
- Vs. Race-directed [Sen, 2008]
  - CD scheduling bounds the # of contexts.
- Vs. Context-bounded [Qadeer, 2005]
  - CD scheduling decides preemption locations with detected conflicts.